**作业1实验报告**

# 概述

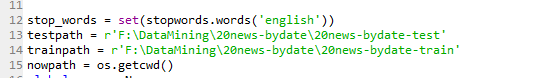
这是DataMining作业1的实验报告，包括了运行环境和使用方法介绍，主要思想的阐述，主要方法的说明，结果的展示和分析，和简单的结论。作业主体是一个完整的knn算法，该算法包括了完整的预处理和预测计算过程，可以以较高的正确率对文档进行分类。

# 环境及使用方法介绍

本程序运行于python3.6环境（或许别的python版本也可），我是用的开发环境是anaconda，编辑器是spyder。

需要的模块包括：numpy库，nltk库，textblob库，scipy库，其中nltk库需要运行nltk.download()将相关文件下载到本地。

下面是使用方法介绍：



第13和14行指定了训练集和测试集的路径，使用者需要把正确的训练集和测试集的路径写入。

在程序的最后有这样两个方法，其中generateVSM()是对训练数据生成对应的训练数据，生成对应的相应的向量空间；



excute(testpath，50)是对测试数据进行预测的方法，第一个参数可以传入任意测试集路径，第二个参数指定的是k值，需要说明的是，算法会对任意k1<=k的k1均做一遍预测并计算准确率，使用者可以方便的对比k取多少时可以获得最好的效果，而且这并不比只做一次k1=k的预测消耗更多的时间复杂度。

# 主要思想介绍

整体来看，程序包括三大步骤:预处理步骤，生成向量空间，进行knn。

1.预处理步骤。主要包括分词，词根还原和去停用词。分词把每篇文章变成一个只包含单词的list，词根还原把动词的时态去除，把名词的单复数等变形去除，去停用词把一些过于泛用的词去除。在这里我使用了textblob这个集成了所有功能的库，停用词表则使用nltk内置的停词表。这样就把一篇文章变成了一个只含有有效词汇的list。

2.生成向量空间。向量空间的每一维度相当于一个词，每篇文章在这个维度上的值是tf\*idf,tf是该词在该文章中的频次，idf是log(总文章数/该词出现的文章数)。于是算法首先要维护一个reverse list，对于每一个单词维护一个dict，里面存储了这个单词在第m篇文章出现的次数n，0<m<=总文章数。所有的单词的dict组成在一起得到这个reverse list。因为词数过大，我选择将idf过小和过大的词过滤去，提高运算速度和精度。由reverse list可以得到整个向量空间，其中文章m每个维度的df=n，idf=log(总文章数/该单词的dict长度)。于是可以生成整个的向量空间。

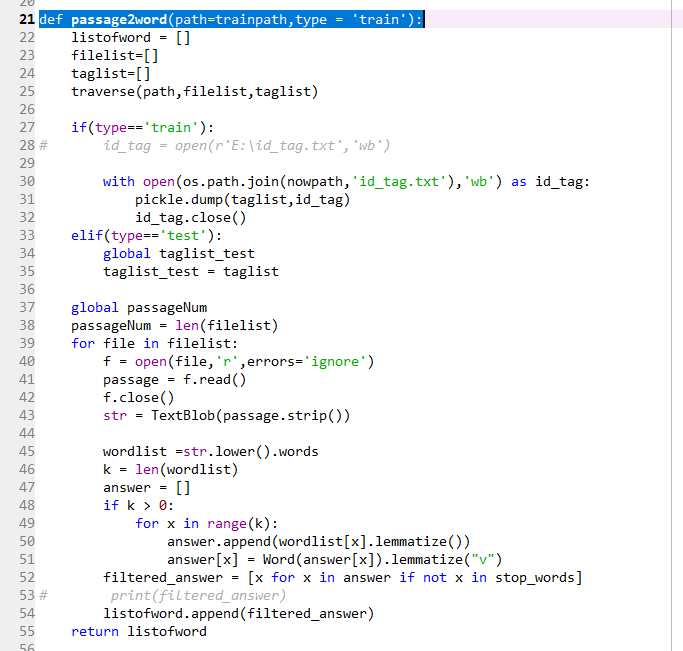
3.knn。距离的计算采用余弦距离，具体可以用矩阵运算得出，在此不一一赘述。运算的矩阵中大部分为稀疏矩阵，因此可以用稀疏矩阵包scipy进行加速，遗憾的是并不是所有需要的操作scipy都给出了支持，自己编写则非常繁琐，因此没法达到理论的最快时间复杂度，但对实际运行时间的加速还是很明显的(加速在十倍以上，加速后对20000维3000篇文章根据50个k值预测50次只需要约20s，不包括预处理时间)。

另外算法为了节约内存空间和作为项目的原则需求，把训练集生成的数据写在了文件中，方便随时读取，但均为二进制文件，若要阅读需要pickle模块读取阅读。

在测试集的处理上，前两个模块基本一致，只需要注意在训练集中没出现的词不需要再考虑，直接删去即可，另外需要一个dict保持训练集和测试集的维度一致。

# 主要方法介绍

1. passage2word(path=trainpath,type = 'train')方法：



该方法运用textblob库对文章进行预处理，第一个参数表示要对哪个路径下的文章进行处理，第二个参数标志是对train集做处理还是对test集做处理，返回值是一个处理过后的文章的词的二维list（wordlist）。

1. 将wordlist转化为reverse list的方法:



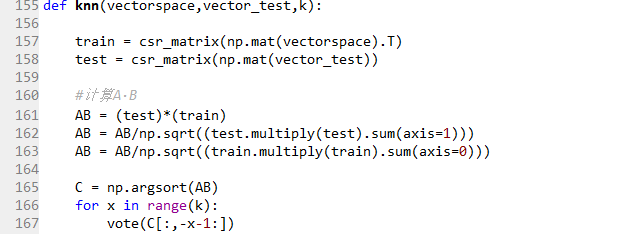
输入是一个wordlist，返回值是一个二维字典reverselist。第二个参数为’no’则代表对测试集合处理。代码主体是计算出字典。

1. 由reverselist生成vsm的方法：



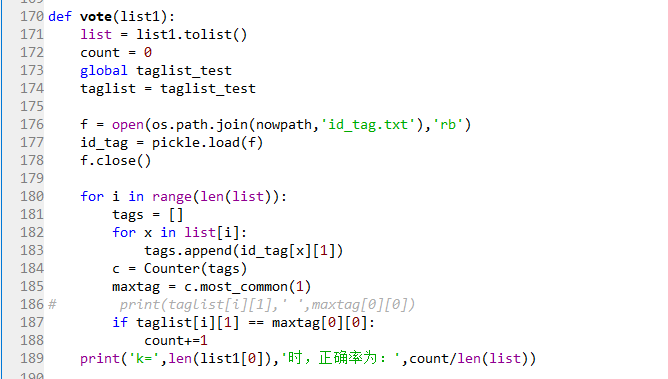
输入是一个reverselist，输出是一个vsm，代码主体是计算tf和idf，另外把维度映射dict和vsm写到了文件中。

1. knn核心方法，计算距离和预测。



其计算过程比较简单，157和158行把矩阵转化为稀疏矩阵，161，162，163行进行对稀疏矩阵进行余弦距离的计算，165行对余弦距离进行排序，166，167行选出前k个最近的文档，调用vote方法统计最合适的tag。

5.投票函数，用于统计最近的k个文档中出现次数最多的tag



其中用了python中自带的counter类，可以方便的返回一个集合中出现次数最多的元素。主要功能是被knn()函数调用，得出文档最终预测的tag，即为代码中的‘maxtag’。

1. 把测试文档转化为vsm的方法：



其中可以调用对train集合的处理方法passage2word()，并且在转化为对应的vsm时注意读取训练集的reverse list文件和维度映射文件，以便保持维度一致。计算idf只需读取reverse list中的idf项即可。

# 结果展示

1. 文档转化为的word list：



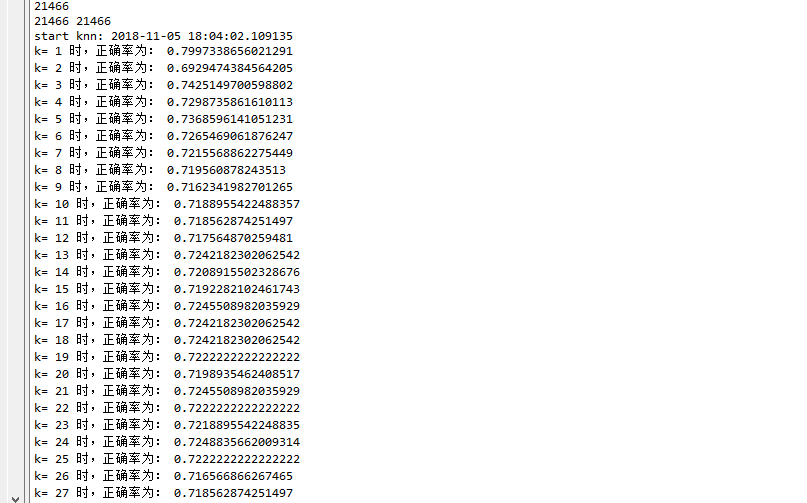
可以看到训练集文档总数为12063。

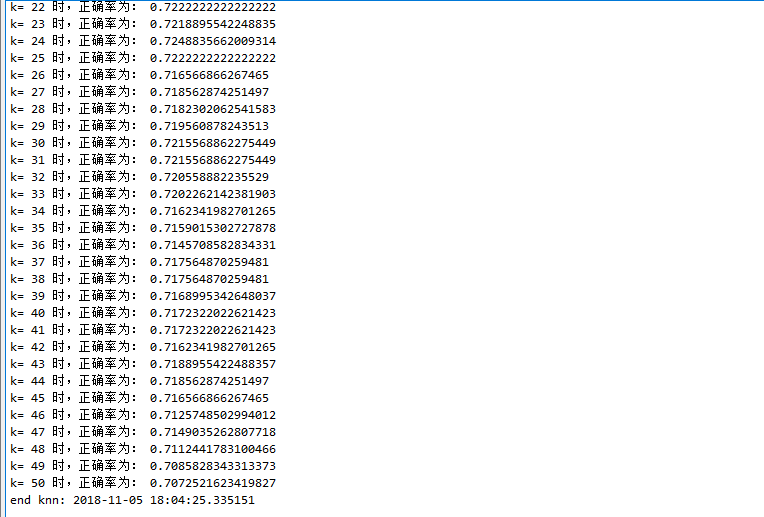
2.训练集的reverse list；



可以看到每个单词里面都是{单词：{文章号，次数}}的形式。

3.运行结果：





可以看到vsm的维数大约在20000左右，正确率在70%到80%之间，预测50次的总时间约为20s。

其余中间结果均可打印，在此不一一展示。

# 六．总结

虽然knn是个较为简单的办法，但是实验起来发现，其效果也是较为不错的。在接近20000篇文档的情况下，分类的正确率在70~80之间，对训练集的预处理大约5~10分钟，对测试集的处理大概3~5分钟，knn核心计算大约只需要20s。当然，80%的准确率对于其他复杂方法是相当低的，但是考虑到方法的简单，knn得到的结果依然是性价比较高的。

通过这次实验，我也学到了很多东西，从第一次接触python，到实现第一个完整的较大的算法，从无到有是最重要的一步，其中也遇到了许多的困难，但克服之后，对python和knn的理解也深了一些。同时也感谢教导我的老师，与我交流学习的同学。总而言之，这是一次非常有意义的课设。s